Java First-Tier: Aplicações

Construção de Interfaces com Swing (II)

Grupo de Linguagens de Programação

Departamento de Informática
PUC-Rio

Lista de Opções

Classe **JList**

- Modela um elemento que apresenta uma lista de opções ao usuário
- Permite a seleção simples (um único elemento), ou múltipla (vários elementos)

Modos de Seleção

- SINGLE_SELECTION
  - Apenas um elemento pode ser selecionado
- SINGLE_INTERVAL_SELECTION
  - Um intervalo contíguo de elementos pode ser selecionado
- MULTIPLE_INTERVAL_SELECTION
  - Não há restrições sobre a seleção (“default”)

Métodos de **JList**

```
JList(ListModel listModel)
JList(Object[] listData)
JList(Vector listData)

void setSelectionMode(int selectionMode)

int getSelectedIndex()
int[] getSelectedIndices()

Object getSelectedValue()
Object[] getSelectedValues()
```


Exemplo de **JList**

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
String[] nomes = {"a", "b", "c", "d", "e", "f"};
JList l = new JList(nomes);
Container cp = f.getContentPane();
cp.add(l);
f.pack();
f.show();
```

Eventos de Seleção

- Eventos de seleção são gerados sempre que a seleção de uma lista é alterada.
- Esses eventos podem ser tratados através da adição de um **ListSelectionListener**.
- A interface **ListSelectionListener** pertence ao pacote javax.swing.event e define apenas um método: **`valueChanged`**

7

Exemplo de **ListSelectionListener**

```
class MeuListener implements ListSelectionListener {
  public void valueChanged(ListSelectionEvent e) {
    if (e.getValueIsAdjusting())
      return;
    JList lista = (JList)e.getSource();
    if (lista.isSelectionEmpty()) {
      … }
    else {
      int index = lista.getSelectedIndex();
      String val = (String)lista.getSelectedValue();
      … }
   }
}
```

8

Barras de Rolagem

- No Swing, ao invés de repetir o código que implementa o uso de barras de rolagem (*scrollbars*) em diferentes elementos, foi criado um elemento de interface cujo único objetivo é fornecer essa funcionalidade
- A classe **JScrollPane** modela esse elemento e a interface **Scrollable** define o que cada elemento deve ter para poder ser tratado por um **JScrollPane**

9

Elementos *Scrollable*

- As classes **JList**, **JTextComponent**, **JTree** e **JTable** implementam a interface **Scrollable** e, portanto, suas instâncias podem ser usadas com **JScrollPane**

10

Exemplo: **JScrollPane** com **JList**

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
String[] nomes = {"a", "b", "c", "d", "e", "f"};
JList l = new JList(nomes);
JScrollPane sp = new JScrollPane(l);
Container cp = f.getContentPane();
cp.add(sp);
f.pack();
f.show();
```

11

Exemplo: **JScrollPane** com **JList**

12

JList: Explorando a Arquitetura MVC

- Como os dados (o modelo) não fazem parte integrante do elemento de interface que os exibe, podemos gerenciá-los em separado
- Por exemplo, é possível exibir um mesmo conjunto de dados em mais de um elemento de interface, simultaneamente
- Também é possível fazer com que o elemento de interface use os dados originais, sem copiá-los

13

Exemplo de Uso

- Suponha que você tem uma lista de nomes muito grande e deseja exibí-la em uma **JList**
- Usando a forma que vimos, esses nomes seriam copiados para dentro da lista
- Para evitar essa replicação, podemos utilizar um modelo próprio, que permitirá à **JList** acessar diretamente a lista de nomes

14

Interface **ListModel**

- Define o modelo usado pela classe **JList**
- Abrange dois aspectos:
  1. o acesso aos dados
  2. o controle da modificação dos dados

15

Métodos de **ListModel**

```
int getSize()

Object getElementAt(int index)

void addListDataListener(
                        ListDataListener l)
void removeListDataListener(
                        ListDataListener l)
```

16

De Volta ao Exemplo

- Imagine que os nomes estão armazenados em um *array* de **String**
- Assumindo que a lista de nomes não é modificada, podemos ignorar o *listener*
- Basta, então, definir uma classe que implemente **ListModel** e utilize o *array* como fonte dos dados

17

Criando um Modelo

```
class ListaDeNomes implements ListModel {
  private String[] nomes;
  ListaDeNomes(String[] nomes) {
    this.nomes = nomes;
  }
  public int getSize() {
    return nomes.length;
  }
  public Object getElementAt(int index) {
    return nomes[index];
  }
  public void addListDataListener(ListDataListener l) {}
  public void removeListDataListener(ListDataListener l) {}
}
```

18

Usando o Modelo

```
JFrame f = new JFrame("Teste");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
String[] nomes = {"a", "b", "c", "d", "e", "f"};
JList l = new JList(new ListaDeNomes(nomes));
Container cp = f.getContentPane();
cp.add(new JScrollPane(l));
f.pack();
f.show();
```

19

Menus

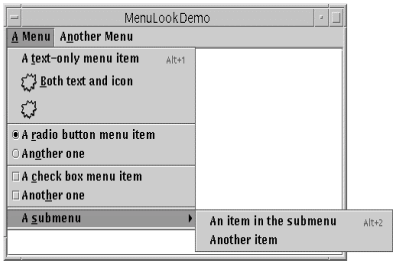
20

Menus

- A biblioteca Swing disponibiliza menus comuns (*pull-down*) e menus *pop-up*
- Um menu pode conter itens de menu e separadores, sendo que os itens podem ter texto e imagem, além de poderem ser marcados (como um *checkbox*)

21

Exemplo:



© The Java ™ Tutorial

22

Hierarquia de Classes

```
JComponent
    ├→ JMenuBar
    ├→ JPopupMenu
    ├→ JSeparator
    │
    └→ JAbstractButton
           └→ JMenuItem
                  ├→ JMenu
                  ├→ JCheckButtonMenuItem
                  └→ JRadioButtonMenuItem
```

23

Classe **JMenuBar**

- Modela uma barra de menu, fixa em uma janela da aplicação
- Na barra de menu pode-se colocar os menus da aplicação, sendo que o menu de ajuda (*help*) possui um tratamento diferenciado

24

Métodos de **JMenuBar**

```
JMenu add(JMenu c)
JMenu getMenu(int index)
int getMenuCount()

void setHelpMenu(JMenu menu)
JMenu getHelpMenu()
```

25

Classe **JMenu**

- Modela um menu que pode ser colocado na barra de menus ou dentro de um outro menu
- Pode conter:
  - itens de menu
  - separadores

26

Métodos de **JMenu**

```
JMenu(String s)

JMenuItem add(String name)
JMenuItem add(JMenuItem menuItem)
JMenuItem insert(JMenuItem i, int p)

void addSeparator()
void insertSeparator(int pos)

void setMnemonic(char mnemonic)
```

27

Classe **JMenuItem**

- Modela um item de menu
- É superclasse de **JMenu**, uma vez que um menu também pode ser um item de menu
- É sub-classe de **JAbstractButton**, logo, um item de menu é um botão

28

Métodos de **JMenuItem**

```
JMenuItem(String text)
JMenuItem(String text, Icon icon)

void setMnemonic(char mnemonic)
void setAccelerator(KeyStroke keyStroke)
void setEnabled(boolean b)

void addActionListener(ActionListener l)
```

29

Exemplo de Menus

```
JFrame f = new JFrame("Teste");
JMenuBar b = new JMenuBar();
JMenu m = b.add(new JMenu("Arquivo"));
m.setMnemonic('a');
JMenuItem i = m.add("Sair");
i.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ev) {
    System.exit(0);
  }});
i.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
                                        KeyEvent.CTRL_MASK));
i.setMnemonic('s');
f.setJMenuBar(b);
f.pack();
f.show();
```

30

Menus de Escolha

- **JMenuItem** possui duas sub-classes que permitem a escolha de opções:
  - **JCheckBoxMenuItem**
  - **JRadioButtonMenuItem**
- Essas classes modelam itens de menu que funcionam da mesma maneira que os botões de escolha que vimos

31

Classe **JPopupMenu**

- Modela um menu *pop-up*, isto é, um menu que pode ser aberto sobre um elemento qualquer de interface, fora da barra de menu
- Assim como um menu comum, um menu *pop-up* pode conter itens de menu e separadores

32

Métodos de **JPopupMenu**

```
JPopupMenu()
JPopupMenu(String label)

JMenuItem add(String name)
JMenuItem add(JMenuItem menuItem)
void addSeparator()

void pack()
void show(Component c, int x, int y)
```

33

Exemplo de **JPopupMenu**

```
JFrame f = new JFrame("Teste");
final JPopupMenu p = new JPopupMenu();
JMenu m = new JMenu("Arquivo");
m.add("Sair");
p.add(m);
f.addMouseListener(new MouseAdapter() {
  public void mousePressed(MouseEvent ev) {
    if (ev.isPopupTrigger())
      p.show((Component)ev.getSource(), ev.getX(), ev.getY());
  }
  public void mouseReleased(MouseEvent ev) {
    if (ev.isPopupTrigger())
      p.show((Component)ev.getSource(), ev.getX(), ev.getY());
  }});
f.pack();
f.show();
```

34

Diálogos Pré-definidos

35

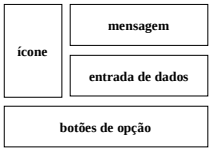
Diálogos Pré-definidos

- O Swing oferece um conjunto de diálogos simples pré-definidos para uso em interações breves com o usuário
  - mensagens de erro, de alerta
  - obtenção de uma confirmação
  - entrada de um único campo de texto
- Esses diálogos são *modais*

36

Classe **JOptionPane**

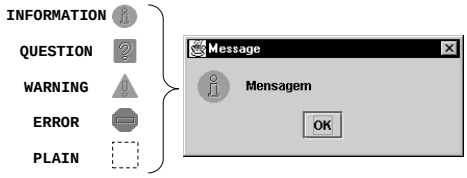
- Métodos estáticos para a criação desses diálogos simples
- Estrutura básica:



37

MessageDialog

- Exibe uma mensagem e aguarda OK do usuário



38

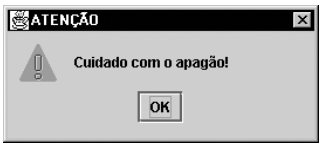
Método **showMessageDialog**

```
void showMessageDialog(Component parentComponent,
                       Object message);
void showMessageDialog(Component parentComponent,
                       Object message,
                       String title,
                       int messageType);
void showMessageDialog(Component parentComponent,
                       Object message,
                       String title,
                       int messageType,
                       Icon icon);
```

39

Exemplo de **MessageDialog**

```
JOptionPane.showMessageDialog(janela,
                              "Cuidado com o apagão!",
                              "ATENÇÃO",
                              JOptionPane.WARNING_MESSAGE);
```



40

ConfirmDialog

- Exibe uma mensagem e obtém uma confirmação (YES/NO, OK/CANCEL)
- Conjuntos de botões de opção ***(optionType)***:
  - **`JOptionPane.YES_NO_OPTION`**
  - **`JOptionPane.YES_NO_CANCEL_OPTION`**
  - **`JOptionPane.OK_CANCEL_OPTION`**

41

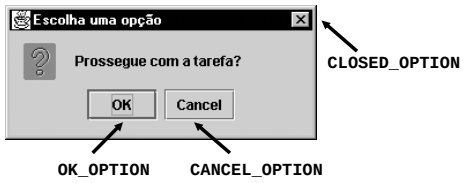
Método **showConfirmDialog**

```
int showConfirmDialog(Component parentComponent,
                      Object message);
int showConfirmDialog(Component parentComponent,
                      Object message,
                      String title,
                      int optionType);
int showConfirmDialog(Component parentComponent,
                      Object message,
                      String title,
                      int optionType
                      int messageType,
                      Icon icon);
```

42

Exemplo de **ConfirmDialog**

```
int resp = JOptionPane.showConfirmDialog(janela,
                            "Prossegue com a tarefa?",
                            "Escolha uma opção",
                            JOptionPane.OK_CANCEL_OPTION);
```



43

InputDialog

- Exibe uma mensagem e obtém um valor de entrada do usuário
  - campo de texto editável
  - *combo box*

44

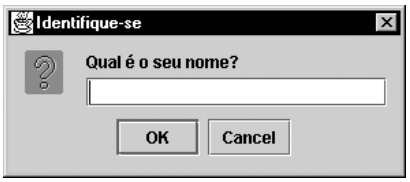
Método **showInputDialog**

```
String showInputDialog(Component parentComponent,
                       Object message);
String showInputDialog(Component parentComponent,
                       Object message,
                       String title,
                       int messageType);
Object showInputDialog(Component parentComponent,
                       Object message,
                       String title,
                       int messageType
                       Icon icon,
                       Object[] selectionValues,
                       Object defaultSelection);
```

45

Exemplo de **InputDialog**

```
String nome = JOptionPane.showInputDialog(janela,
                            "Qual é o seu nome?",
                            "Identifique-se",
                            JOptionPane.QUESTION_MESSAGE);
```



46

OptionDialog

- Exibe uma mensagem (ou objeto) e obtém uma opção escolhida pelo usuário
- O número de botões e seus textos são configuráveis
- A opção *default* é configurável

47

Método **showOptionDialog**

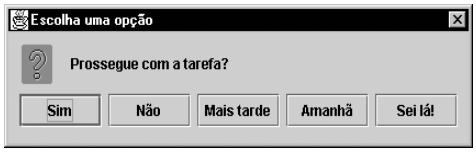
```
int showOptionDialog(Component parentComponent,
                     Object message,
                     String title,
                     int optionType
                     int messageType,
                     Icon icon,
                     Object[] options,
                     Object initialValue);
```

48

Exemplo de **OptionDialog**

```
Object[] opções = {"Sim", "Não", "Mais Tarde",
                   "Amanhã", "Sei lá!"};
int resp = JOptionPane.showOptionDialog(janela,
                           "Prossegue com a tarefa?",
                           "Escolha uma opção",
                           JOptionPane.DEFAULT_OPTION,
                           JOptionPane.QUESTION_MESSAGE,
                           null,
                           opções,
                           opções[0]);
```


Exemplo de **OptionDialog**



Classe **JFileChooser**

- É comum uma aplicação abrir e salvar arquivos
- A classe **JFileChooser** implementa uma caixa de diálogo que permite que o usuário navegue pelo sistema de arquivos
  - semelhante às usadas por aplicações “nativas”

Diálogo de Seleção de Arquivos

Métodos de **JFileChooser**

```
void setCurrentDirectory(File dir)
void setSelectedFile(File file)
void setMultiSelectionEnabled(boolean b)
void setFileFilter(FileFilter filter)

int showOpenDialog(Component parent)
int showSaveDialog(Component parent)
int showDialog(Component parent,
               String approveButtonText)

File getSelectedFile()
File[] getSelectedFiles()
```


Exemplo de **JFileChooser**

```
JFileChooser chooser = new JFileChooser();
chooser.setCurrentDirectory(new File("c:\\jdk1.3"));
int res = chooser.showOpenDialog(janela);
if (res == JFileChooser.APPROVE_OPTION) {
   File file = chooser.getSelectedFile();
   System.out.println(file.getName());
}
```

Classe **FileFilter**

- Utilizada para a implementação de filtros que permitem restringir os tipos de arquivos exibidos em um diálogo de seleção
  - **FileFilter** é uma classe abstrata
- A subclasse de **FileFilter** deverá implementar os métodos:

```
boolean accept(File file)
String getDescription()
```

55

Exemplo de **FileFilter**

```
import javax.swing.filechooser.FileFilter;

public class GifFilter extends FileFilter
{
  public boolean accept(File f) {
    return f.getName().toLowerCase().endsWith(".gif")
           || f.isDirectory();
  }
  public String getDescription() {
    return "Arquivos GIF";
  }
}
```

56

Classe **JTree**

- Componente que exibe uma estrutura de dados hierárquica (árvore)
- Segue o padrão MVC: os dados a serem exibidos são obtidos de um modelo (**TreeModel**)
  - o modelo a ser utilizado é fornecido no construtor do objeto **JTree**

57

Árvore

58

Terminologia

- Uma árvore é composta de nós
  - um nó ou é uma **folha** ou possui nós filhos
  - todo nó, com exceção da raiz, tem exatamente um nó pai
  - toda árvore tem exatamente um nó raiz
- Tipicamente, o usuário pode **expandir** ou ***colapsar*** nós, tornando seus filhos, respectivamente, visíveis ou invisíveis

59

Exemplos

60

Interface **TreeModel**

- Define um modelo de dados adequado para um JTree
- Pertence ao pacote **javax.swing.tree**
- O Swing oferece uma implementação dessa interface: a classe **DefaultTreeModel**
  - modelo de árvore que utiliza **TreeNodes**

61

Métodos de **DefaultTreeModel**

```
DefaultTreeModel(TreeNode root)

Object getRoot()
int getChildCount(Object parent)
Object getChild(Object parent, int index)

void setAsksAllowsChildren(boolean newValue)
void insertNodeInto(MutableTreeNode child,
                    MutableTreeNode parent,
                    int index)
void removeNodeFromParent(MutableTreeNode node)

void addTreeModelListener(TreeModelListener l)
```

62

Interface **MutableTreeNode**

- É uma subinterface de **TreeNode**
- Modela um nó que pode ser modificado
  - adição/remoção de filhos
  - modificação do conteúdo armazenado no nó (“user object”)
- O Swing oferece uma implementação dessa interface: a classe **DefaultMutableTreeNode**

63

Métodos de **DefaultMutableTreeNode**

```
DefaultMutableTreeNode(Object userObject)
DefaultMutableTreeNode(Object userObject,
                       boolean allowsChildren)
void add(MutableTreeNode child)
void remove(MutableTreeNode child)

Object getUserObject()
void setUserObject(Object userObject)
String toString()
void setAllowsChildren(boolean allows)
boolean isLeaf()
Enumeration children()
```

64

Criando um JTree

```
DefaultMutableTreeNode mundo =
      new DefaultMutableTreeNode (“Mundo”);
DefaultMutableTreeNode europa =
      new DefaultMutableTreeNode (“Europa”);
DefaultMutableTreeNode americas =
      new DefaultMutableTreeNode (“Américas”);
mundo.add(europa);
mundo.add(americas);
…
JTree arvore = new JTree(new DefaultTreeModel(mundo));
```

65

Modos de Seleção

- O modo de seleção de um JTree é configurado (e gerenciado) por um “modelo de seleção” (**TreeSelectionModel)**
- Modos disponíveis:
  - SINGLE_TREE_SELECTION
  - CONTIGUOS_TREE_SELECTION
  - DISCONTIGUOUS_TREE_SELECTION

66

Configurando o modo de seleção

```
JTree arvore = new JTree(raiz);
int modo = TreeSelectionModel.SINGLE_TREE_SELECTION;
TreeSelectionModel tsm = arvore.getSelectionModel();
tsm.setSelectionMode(modo);
```

67

Obtendo a seleção corrente

```
JTree arvore = new JTree(new DefaultTreeModel(raiz));
 …
TreePath path = arvore.getSelectionPath()
if (path != null) {
 DefaultMutableTreeNode selNode =
(DefaultMutableTreeNode)path.getLastPathComponent();
String selValue = (String)selNode.getUserObject();
 …
}
```

68

Eventos de Seleção

- Eventos de seleção são gerados sempre que a seleção de uma árvore é alterada.
- Esses eventos podem ser tratados através da adição de um **TreeSelectionListener**.
- A interface **TreeSelectionListener** pertence ao pacote javax.swing.event e define apenas um método: **`valueChanged`**

69